Begin Microsoft Small Basic: Hoofdstuk 3: Uw eerste kleine basisprogramma

Inhoudsopgave

Bekijken en bekijken

In de eerste twee klassen heb je geleerd over de structuur van een Small Basic-programma, enkele regels voor het typen van code en het uitvoeren van een Small Basic-programma. Heb je ideeën over programma's die je zou willen bouwen met Small Basic? Als dat zo is, geweldig.

Dit hoofdstuk is een bewerking van

Vanaf deze les begin je met het ontwikkelen van je eigen programmeervaardigheden. In elke les die nog moet komen, leer je een aantal nieuwe functies van de Small Basic-taal. In deze les schrijf je je eerste Small Basic-programma. Om dit te doen, moet u eerst enkele van de basiscomponenten van de kleine basistaal leren kennen. Je leert over variabelen, toewijzingsinstructies en enkele eenvoudige operators.

Een klein basisprogramma maken

Bedenk uit klasse 2 dat een **Small Basic-verklaring** iets doet. In het **welkomstvoorbeeld** zagen we een verklaring waarin wat informatie werd afgedrukt ("Welkom bij Beginning Small Basic!"). Elk programma dat we in deze klas bouwen, bestaat uit veel kleine basisinstructies die de computer kan verwerken. Het maken van een computerprogramma met Small Basic (of een andere taal) is een eenvoudig proces. U hebt een bepaalde taak waarvan u wilt dat de computer deze voor u doet. U vertelt de computer in een logische, procedurele reeks stappen hoe u die taak moet uitvoeren.

het boek *BEGINNING Microsoft Small Basic* van Philip Conrod en Lou Tylee.

Om dit boek in zijn geheel te kopen, zie de Computer Science For Kids website .

Het is relatief eenvoudig om oplossingsstappen voor een probleem in onze taal uit te schrijven (Engels, in deze notities). Het moeilijke deel is dat je met de computer in zijn eigen taal moet praten. Het zou mooi zijn als we gewoon konden schrijven"Hé computer, hier zijn twee getallen - tel ze bij elkaar op en vertel me de som." Een mens kan deze instructies begrijpen, maar een computer niet. Waarom? Ten eerste moet de computer worden verteld hoe taken in zeer specifieke, logische stappen moeten worden uitgevoerd. Voor dit kleine voorbeeld van een toevoeging zijn de stappen:

1. Geef een waarde aan het eerste getal.
2. Geef een waarde aan het tweede getal.
3. Voeg het eerste getal toe aan het tweede getal, wat resulteert in de som, een derde getal.
4. Vertel me de som.

Vervolgens moeten we met de computer in zijn eigen taal praten. We vertalen elke oplossingsstap naar een statement (of statements) in de taal van de computer. En in deze cursus is de taal van de computer **Small Basic**. Om de computer te kunnen vertellen hoe hij een taak moet uitvoeren, moet u een grondig begrip hebben van de Small Basic-taal. Uw begrip van Small Basic stelt u in staat om uw programmeerstappen te vertalen naar een taal die de computer kan begrijpen.

Een ander ding om te onthouden als je kleine basisprogramma's schrijft, is dat je logisch en nauwkeurig moet zijn. Een computer zal uw instructies volgen - zelfs als ze verkeerd zijn! Dus, terwijl je Small Basic leert, zullen we de noodzaak benadrukken om precies te zijn. Zodra u exacte en logische Small Basic-code schrijft, is de computer erg goed en snel in het uitvoeren van zijn werk. En het kan een aantal behoorlijk verbazingwekkende dingen doen. Laten we eens kijken naar een paar andere voorbeelden van het uitschrijven van een programmeertaak als een reeks stappen om enkele dingen te illustreren die een computer kan doen.

Wat als het lokale schoolhoofd je vraagt om de testscores van de 352 leerlingen in de school te gemiddelden? Deze stappen zijn:

1. Bepaal de score van elke student.
2. Tel de 352 scores bij elkaar op om een som te krijgen.
3. Deel de som door 352 om de gemiddelde waarde te krijgen.
4. Vertel de opdrachtgever het gemiddelde.

Not too hard, huh? Notice here that the second step can be further broken down into smaller steps. To add up 352 scores, you would:

1. Start with the first score.
2. Add in the second score, then the third score, then the fourth score, etc.
3. Stop when all scores have been added.

In these steps, the computer would do the same task (adding a number) 352 times. Computers are very good at repeating tasks – we will see that this process of repetition is called **looping**. You will build code for this example in Class 7.

Computers are also very good at playing games with you (that's why video games are so popular). Have you ever played the card game "War?" You and another player take a card from a standard playing deck. Whoever has the 'highest' card wins the other player's card. You then each get another card and continue the comparison process until you run out of cards. Whoever has the most cards once the game stops is declared the winner. Playing this game would require steps similar to these:

1. Shuffle a deck of cards.
2. Give a card to the first player.
3. Give a card to the second player.
4. Determine which card is higher and declare a winner.
5. Repeat the process of giving cards to players until you are out of cards.

Things are a bit more complicated here, but the computer is up to the task. The first step requires the computer to shuffle a deck of cards. How do you tell a computer how to do this? Well, before this course is over, you will know how. For now, just know that it's a series of several programming steps. We will put the Small Basic program for such a specific task in its own area called a **subroutine**. This makes the program a little easier to follow and also allows use this code in other programs. Notice Step 4 requires the computer to make a **decision** – determining which card is higher. Computers are very good at making decisions. Finally, Step 5 asks us to repeat the handing out of cards – another example of **looping**. You will also build this program in Class 7.

If all of these concepts are not clear at the moment, that's okay. They will become clearer as you progress through this course. I just wanted you to have some idea of what you can do with Small Basic programs. Just remember, for every Small Basic program you create, it is best to first write down a series of logical steps you

want the computer to follow in performing the tasks needed by your program. Then, converting those steps into the Small Basic language will give you your Small Basic program – it's really that simple. This class begins instruction in the elements of Small Basic. And, in subsequent classes, you learn more and more Small Basic, adding to your Small Basic vocabulary. We'll start slow. By the end of this course, you should be pretty good at "talking Small Basic."

Small Basic - The First Lesson

At long last, we are ready to get into the heart of a Small Basic program - the Small Basic language. In this class, we will discuss variables (name and type), assignments, arithmetic operations, and techniques for working with a particular type of variable called strings. In each subsequent class in this course, you will learn something new about the Small Basic language.

Variables

All computer programs work with information of one kind or another. Numbers, text, dates and pictures are typical types of information they work with. Computer programs need places to store this information while working with it. What if we need to know how much ten bananas cost if they are 25 cents each? We would need a place to store the number of bananas, the cost of each banana, and the result of multiplying these two numbers together. To store such information, we use something called **variables**. They are called variables because the information stored there can change, or vary, during program execution. Variables are the primary method for moving information around in a Small Basic program. And, certain rules must be followed in the use of variables.

Variable Names

You must **name** every variable you use in your program. Rules for naming variables are:

- Can only use letters, numbers, and the underscore (_) character (though the underscore character is rarely used).
- The first character must be a letter. It is customary, though not required, in Small Basic that this first letter be upper case
- You cannot use a word reserved by Small Basic (for example, you can't have a variable named **WriteLine** or one named **TextWindow**).

If a variable name consists of more than one word, the words are joined together, and each word after the first begins with an uppercase letter

The most important rule is to use variable names that are meaningful. You should be able to identify the information stored in a variable by looking at its name. As an example, in our banana buying example, good names would be:

Quantity	**Variable Name**
Cost of each banana	BananaCost
Number of bananas purchased	NumberBananas
Cost of all bananas	TotalBananaCost

As mentioned in an earlier class, the Small Basic language is not case sensitive. This means the names **BananaCost** and **bananacost** refer to the same variable. Try to be consistent in how you write variable names. And make sure you assign unique names to each variable. A nice feature of the Small Basic intellisense feature is that as you add variables to your program, the variable names are added to the list of choices available in the intellisense drop-down menu.

Variable Types

We need to know the **type** of information stored by each variable. Does it contain a number? Does the number have a decimal point? Does it just contain text information? Let's look at some variable types.

The first variable type is the **integer** type. This type of variable is used to represent whole, non-decimal, numbers. Examples of such numbers are:

1 -20 4000

Notice you write 4,000 as 4000 in Small Basic – we can't use commas in large numbers. In our banana example, **NumberBananas** would an **integer** type variable.

What if the variable you want to use will have decimal points. In this course, such variables will be of **floating** type (the decimal point being the thing that "floats"). All you need to know about floating type variables is that they are numbers with decimal points. Examples of such numbers:

-1.25 3.14159 22.7

In our banana example, the variables **BananaCost** and **TotalBananaCost** would be **floating** type variables.

The next variable type is a **string** variable. A **string** variable is just that – one that stores a string (list) of various characters. A string can be a name, a string of numbers, a sentence, a paragraph, any characters at all. And, many times, a string will contain no characters at all (an empty string). We will use lots of strings in Small Basic, so it's something you should become familiar with. Strings are always enclosed in quotes ("). Examples of strings:

"I am a Small Basic programmer" "012345" "Title Author"

One last variable type is the **boolean** variable. It takes its name from a famous mathematician (Boole). It can have one of two values: **true** or **false**. Many languages have **true** and **false** as reserved words for use with Boolean variables. Small Basic does not have such reserved words. In Small Basic, we will use string values of "**true**" and "**false**" to provide a representation for boolean variables. We will see that such variables are at the heart of the computer's decision making capability. If wanted to know if a banana was rotten, we could name a boolean variable **IsBananaRotten**. If this was "true", the banana is indeed rotten.

With all the different variable types, we need to be careful not to improperly mix types. We can only do mathematical operations on numbers (integer and floating types). String types must only work with other string types. Boolean types are used for decisions.

Assignment Statement

The simplest, and most widely used, statement in Small Basic is the **assignment** statement. Such a statement appears as:

VariableName = VariableValue

Note that only a single variable can be on the left side of the assignment operator (**=**). Some simple assignment examples using our "banana" variables:

```
1. NumberBananas = 22
2. BananaCost = 0.27
3. IsBananaRotten = "false"
4. MyBananaDescription = "Yes, we have no bananas!"
```

The actual values assigned to variables here are called **literals**, since they literally show you their values.

You may recognize the assignment operator as the equal sign used in arithmetic, but it's not called an equal sign in computer programming. Why is that? Actually, the right side (**VariableValue** in this example) of the assignment operator is not limited to literals. Any legal Small Basic expression, with any number of variables or other values, can be on the right side of the operator. In such a case, Small Basic computes **VariableValue** first, then assigns that result to **VariableName**. This is an important programming concept to remember – "evaulate the right side, assign to the left side." Let's start looking at some operators that help in evaluating Small Basic expressions.

Arithmetic Operators

One thing computer programs are very good at is doing arithmetic. They can add, subtract, multiply, and divide numbers very quickly. We need to know how to make our Small Basic programs do arithmetic. There are four **arithmetic operators** we will use from the Small Basic language.

Addition is done using the plus (**+**) sign and **subtraction** is done using the minus (**-**) sign. Simple examples are:

Operation	Example	Result
Addition	7 + 2	9
Addition	3 + 8	11
Subtraction	6 - 4	2
Subtraction	11 - 7	4

Multiplication is done using the asterisk (*****) and **division** is done using the slash (**/**). Simple examples are:

Operation	Example	Result
Multiplication	8 * 4	32
Multiplication	2 * 12	24
Division	12 / 2	6
Division	42 / 6	7

I'm sure you've done addition, subtraction, multiplication, and division before and understand how each operation works.

What happens if an assignment statement contains more than one arithmetic operator? Does it make any difference? Look at this example:

```
7 + 3 * 4
```

What's the answer? Well, it depends. If you work left to right and add 7 and 3 first, then multiply by 4, the answer is 40. If you multiply 3 times 4 first, then add 7, the answer is 19. Confusing? Well, yes. But, Small Basic takes away the possibility of such confusion by having rules of **precedence**. This means there is a specific order in which arithmetic operations will be performed. That order is:

1. Multiplication (*) and division (/)
2. Addition (+) and subtraction (-)

So, in an assignment statement, all multiplications and divisions are done first, then additions and subtractions. In our example (7 + 3 * 4), we see the multiplication will be done before the addition, so the answer provided by Small Basic would be 19.

If two operators have the same precedence level, for example, multiplication and division, the operations are done left to right in the assignment statement. For example:

```
24 / 2 * 3
```

The division (24 / 2) is done first yielding a 12, then the multiplication (12 * 3), so the answer is 36. But what if we want to do the multiplication before the division - can that be done? Yes - using the Small Basic **grouping operators** - parentheses **()**. By using parentheses in an assignment statement, you force operations within the parentheses to be done first. So, if we rewrite our example as:

```
24 / (2 * 3)
```

the multiplication (2 * 3) will be done first yielding 6, then the division (24 / 6), yielding the desired result of 4. You can use as many parentheses as you want, but make sure they are always in pairs - every left parenthesis needs a right parenthesis. If you nest parentheses, that is have one set inside another, evaluation will start with the innermost set of parentheses and move outward. For example, look at:

```
((2 + 4) * 6) + 7
```

The addition of 2 and 4 is done first, yielding a 6, which is multiplied by 6, yielding 36. This result is then added to 7, with the final answer being 43. You might also want to use parentheses even if they don't change precedence. Many times, they are used just to clarify what is going on in an assignment statement.

As you improve your programming skills, make sure you know how each of the arithmetic operators work, what the precedence order is, and how to use parentheses. Always double-check your assignment statements to make sure they are providing the results you want.

Some examples of Small Basic assignment statements with arithmetic operators:

```
1. TotalBananaCost = NumberBananas * BananaCost
2. NumberOfWeeks = NumberOfDays / 7
3. AverageScore = (Score1 + Score2 + Score3) / 3.0
```

Notice a couple of things here. First, notice the parentheses in the **AverageScore** calculation forces Small Basic to add the three scores before dividing by 3. Also, notice the use of "white space," spaces separating operators from variables. This is a common practice in Small Basic that helps code be more readable. We'll see lots and lots of examples of assignment statements as we build programs in this course.

String Concatenation

We can apply arithmetic operators to numerical variables. String variables can also be operated on. Many times in Small Basic programs, you want to take a string variable from one place and 'tack it on the end' of another string. The fancy word for this is **string concatenation**. The concatenation operator is a plus sign (**+**) and it is easy to use. As an example:

```
NewString = "Beginning Small Basic " + "is Fun!"
```

After this statement, the string variable **NewString** will have the value "Beginning Small Basic is Fun!".

Notice the string concatenation operator is identical to the addition operator. We always need to insure there is no confusion when using both. String variables are a big part of Small Basic. As you develop as a programmer, you need to become comfortable with strings and working with them.

Comments

You should always follow proper programming rules when writing your Small Basic code. One such rule is to properly comment your code. You can place non-executable statements (ignored by the computer) in your code that explain what you are doing. These **comments** can be an aid in understanding your code. They also make future changes to your code much easier.

To place a comment in your code, use the comment symbol, a single apostrophe (**'**). Anything written after the comment symbol will be ignored by the computer. You can have a comment take up a complete line of Small Basic code, like this:

```
1. ' Set number of bananas
2. NumberBananas = 14
```

Or, you can place the comment on the same line as the assignment statement:

```
NumberBananas = 14 ' Set number of bananas
```

You, as the programmer, should decide how much you want to comment your code. We will try in the programs provided in this course to provide adequate comments.

Program Output

You're almost ready to create your first Small Basic program. But, we need one more thing. We have ways to name variables and ways to do math with them, but once we have results, how can those results be displayed? In this class, we will use the method seen in our little Welcome program, the Small Basic **WriteLine** method that works with the **TextWindow** object. What this method does is print a string result on a single line:

```
TextWindow.WriteLine(StringValue)
```

In this expression, **StringValue** could be a string variable that has been evaluated somewhere (perhaps using the concatenation operator) or a literal (an actual value). In the Welcome example, we used a literal:

```
TextWindow.WriteLine("Welcome to Beginning Small Basic!")
```

And saw that **Welcome to Beginning Small Basic!** was output to the text window.

What if you want to output numeric information? It's really quite easy. The **WriteLine** method will automatically convert a numeric value to a string for output purposes. For example, look at this little code segment:

```
1. NumberBananas = 45
2. TextWindow.WriteLine(NumberBananas)
```

If you run this code, a **45** will appear on the output screen. Go ahead and start Small Basic and try it. This is one fun thing about Small Basic. It is an easy environment to try different ideas.

I started a new program in Small Basic and typed these lines in the editor:

When I run this code, I see (I resized the window):

You can also combine text information with numeric information using the concatenation operator. For example, this code:

```
1. NumberBananas = 45
2. TextWindow.WriteLine("Number of Bananas is " + NumberBananas)
```

will print **Number of Bananas is 45** on the output screen:

The numeric data (**NumberOfBananas**) is converted to a string before it is concatenated with the text data.

So, it's pretty easy to output text and numeric information. Be aware one slight problem could occasionally arise though. Recall the concatenation operator is identical to the arithmetic addition operator. Look at this little segment of code:

```
1. NumberBananas = 32
2. NumberApples = 22
3. TextWindow.WriteLine("Pieces of fruit " + NumberBananas + NumberApples)
```

You might think you are printing out the total number of fruit (numberBananas + numberApples = 54) with this statement. However, if you run this code, you will get **Pieces of fruit 3222**:

What happens is that Small Basic converts both pieces of numeric data to a string before the addition can be done. Then, the plus sign separating them acts as a concatenation operator yielding the 3222. To print the sum, we need to force the numeric addition by using parentheses:

```
1. NumberBananas = 32
2. NumberApples = 22
3. TextWindow.WriteLine("Pieces of fruit " + (NumberBananas + NumberApples))
```

In this case, the two numeric values are summed before being converted to a string and you will obtain the desired output of **Pieces of fruit 54**:

So, we see the **WriteLine** method offers an easy-to-use way to output both text and numeric information, but it must be used correctly.

Notice one other thing about this example. The last line of code looks like it's two lines long! This is solely because of the word wrap feature of the word processor being used. In an actual Small Basic program, this line will appear as, and should be typed as, one single line. Always be aware of this possibility when reading these notes. Let's build a program.

Program – Sub Sandwich Party

Your family has decided to have a party. Two very long submarine sandwiches are being delivered and it is your job to figure out how much each person can eat. Sure, you could do this with a calculator, but let's use Small Basic!! This program is saved in the **Sandwich** folder in the course programs folder (**\BeginSB\BSB Code**).

Program Design

Assume you know the length of each submarine sandwich. To make the cutting easy, we will say that each person will get a whole number of inches (or centimeters) of sandwich (no decimals). With this information, you can compute how many people can be fed from each sandwich. If the total number is more than the people you have in your family, everyone eats and things are good. If not, you may have to make adjustments. The program steps would be:

1. Set a value for the number of inches a person can eat.
2. Determine length of both sandwiches.
3. Determine how many people can eat from each sandwich.
4. Increase or decrease the number of inches until the entire family can eat.

Let's translate each of these steps into Small Basic code as we build the program. Since this is your first program, we'll review many steps (creating a new program) and we'll type and discuss the code one or two lines at a time.

Program Development

Start **Small Basic**. Click the **New Program** button in the toolbar. A blank editor will appear. Immediately save the program as **Sandwich** in a folder of your choice.

First, type the following header information as a multi-line comment and add a title to the text window:

```
1. '
2. ' Sub Sandwich Program
3. ' Beginning Small Basic
4. '
5. TextWindow.Title = "Sub Sandwich Party"
```

We will use five variables in this program: one for how much each person can eat (**InchesPerPerson**), two for the sandwich lengths (**LengthSandwich1**, **LengthSandwich2**), and two for how many people can eat from each sandwich (**Eaters1**, **Eaters2**). These will all be numeric variables. Values for Eaters1 and Eaters2 must be whole numbers since you can't have a fraction of a family member. Set values for some of the variables (also include a comment about what you are doing):

```
1. ' set values
2. InchesPerPerson = 5
3. LengthSandwich1 = 114
4. LengthSandwich2 = 93
```

These are just values we made up, you can use anything you like. Notice we assume each person can eat 5 inches of sandwich.

Next, we compute how many people can eat from each sandwich using simple division:

```
1. ' determine how many people can eat each sandwich
2. Eaters1 = Math.Floor(LengthSandwich1 / InchesPerPerson)
3. Eaters2 = Math.Floor(LengthSandwich2 / InchesPerPerson)
```

We hebben iets gebruikt dat nog niet is gezien - de **Math.Floor-functie**. Small Basic biedt een **wiskundebibliotheek** die een groot aantal functies bevat voor ons gebruik (we zullen er in de volgende les meer van bekijken). De functie Math.Floor retourneert het hele getalgedeelte van een decimaal getal - in dit geval verzekert het een geheel getal dat persoon eet.

Merk ook op wanneer u begint met het typen van de line computing Eaters1, de intellisense-vervolgkeuzelijst bevat nu toegevoegde variabelen zoals **LengthSandwich1**:

Geef de resultaten weer met de **writeline-methode**:

```
1. ' write results
2. TextWindow.WriteLine("Letting each person eat " + InchesPerPerson + " inches")
3. TextWindow.WriteLine((Eaters1 + Eaters2) + " people can eat these two sandwiches!")
```

Merk op hoe elk van de aaneenschakelingen van de tekenreeksen werkt. Merk ook op dat we het aantal mensen bij elkaar optellen voordat we het afdrukken.

De voltooide code in Small Basic zou er als volgt uit moeten zien:

```
1. '
2. ' Sub Sandwich Program
3. ' Beginning Small Basic
4. ' TextWindow.Title = "Sub Sandwich Party"
5. ' set values
6. InchesPerPerson = 5
7. LengthSandwich1 = 114
8. LengthSandwich2 = 93
9. ' determine how many people can eat each sandwich
10. Eaters1 = Math.Floor(LengthSandwich1 / InchesPerPerson)
11. Eaters2 = Math.Floor(LengthSandwich2 / InchesPerPerson)
12. ' write results' write results
13. TextWindow.WriteLine("Letting each person eat " + InchesPerPerson + " inches")
14. TextWindow.WriteLine((Eaters1 + Eaters2) + " people can eat these two sandwiches!")
```

Controleer nogmaals of elke regel correct is getypt.

Voer het programma uit

Sla uw programma op (klik op de werkbalkknop **Opslaan**). Voer uw programma uit door op de werkbalkknop **Uitvoeren** te klikken of door op <**F5**> te drukken. Als het programma niet wordt uitgevoerd, worden eventuele fouten onder de editor weergegeven. Controleer nogmaals of uw code exact is - geen spelfouten, geen ontbrekende aanhalingstekens, geen ontbrekende interpunctie. Wanneer het wordt uitgevoerd, moet het tekstvenster dit weergeven:

Dit zegt dat 40 mensen kunnen eten van deze specifieke set broodjes. Gefeliciteerd - je hebt je allereerste Small Basic-programma geschreven!!

Andere dingen om te proberen

Voor elk programma in deze cursus bieden we suggesties voor wijzigingen die u kunt aanbrengen en proberen. In deze bovenstaande run zagen we dat 40 mensen kunnen eten. Wat als je meer of minder moet voeren? Pas de variabele **InchesPerPerson** aan en bepaal het aantal mensen dat voor elke waarde kan eten. Na elke aanpassing moet u het programma opnieuw uitvoeren. Stel dat de broodjes zoveel per centimeter kosten. Pas het programma aan zodat het ook de kosten van de broodjes berekent. Bepaal hoeveel elke persoon zou moeten bijdragen om zijn lunch te betalen. Probeer het eens!

Omdat we van elke persoon eisen dat hij een heel aantal centimeters eet, kunnen er restjes in elke boterham zitten. Kun je erachter komen hoe je dit bedrag kunt berekenen? Het is een nette kleine toepassing van de functie **Rest** (retourneert de rest wanneer twee hele getallen worden gedeeld) in de **wiskundebibliotheek**. Er zijn slechts een paar codewijzigingen. Een nieuwe variabele **InchesLeftOver** wordt gebruikt om het resterende bedrag te berekenen. Nu, de code die die waarde berekent:

```
' compute leftovers

InchesLeftOver = Math.Remainder(LengthSandwich1, InchesPerPerson) + Math.Remainder(L

TextWindow.WriteLine("There are " + InchesLeftOver + " inches left over.")
```

Voeg deze code toe aan uw programma en voer het opnieuw uit. Zie je dat er in totaal nog 7 centimeter over is?

Kun je zien waarom we bij het berekenen van **InchesLeftOver** gewoon niet beide sandwiches aan elkaar toevoegen voordat we de restoperator gebruiken?

Samenvatting

Nogmaals, felicitaties zijn verschuldigd voor het voltooien van je eerste Small Basic-programma. Je hebt veel geleerd over de Small Basic statements en opdrachten en hoe je een beetje rekenwerk kunt doen. U moet vertrouwd zijn met het starten van een nieuw programma met Small Basic. In de volgende lessen zullen we iets meer Small Basic leren en steeds gedetailleerdere Small Basic-programma's schrijven.

Volgende hoofdstuk > >